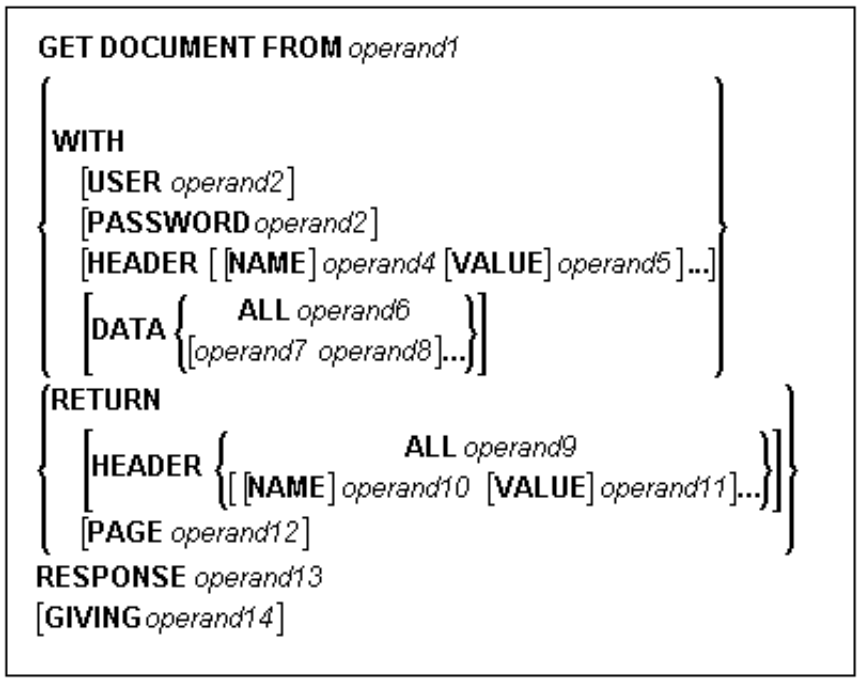
REQUEST DOCUMENT

Note:
This statement is only available under Windows.



Operand	Possible Structure					Possible Formats										Referencing Permitted	Dynamic Definition
Operand1	C	S				A										no	yes
Operand2	C	S				A										no	yes
Operand3	C	S				A										no	yes
Operand4	C	S				A										no	yes
Operand5	C	S				A	N	P	I	F		D	T	L		no	yes
Operand6	C	S				A	N	P	I	F	B	D	T	L		no	yes
Operand7	C	S				A										no	yes
Operand8	C	S				A	N	P	I	F		D	T	L		no	yes
Operand9		S				A	N	P	I	F	B	D	T	L		no	yes
Operand10	C	S				A										no	yes
Operand11		S				A	N	P	I	F	B	D	T	L		no	yes
Operand12		S				A					B					no	yes
Operand13		S				A										no	yes
Operand14		S							I							no	no

Function

The REQUEST DOCUMENT statement gives you the means to access an external system.

Restrictions for Cookies

Under the HTTP Protocol, a server uses cookies to maintain state information on the client workstation.

The implementation of Request Document on Windows uses the internet option settings. This means that, depending on the security settings, Cookies will be handled.

If the setting is set to "Disabled", no cookies will be sent, even if a COOKIE header (operand 4/5) is sent.

For Server environments, do not use the setting "Prompt". This setting leads to a "hanging" server, because no client will be able to answer the prompt.

In the Windows environment, cookies are handled automatically by the Windows API. This means that, if cookies are enabled in the browser, all incoming cookies will be saved and sent automatically with the next request.

operand1

Operand1 is the URL to access a document.

The information below is only valid if operand1 begins with "http://" or "https://".

operand2

Operand2 is the name of the user that is used for the request.

operand3

Operand3 is the password of the user that is used for the request.

operand4/5

Note:
Operand4 and operand5 can only be used in conjunction with each other.

Restrictions

Header Names for Operand4

Header names are not allowed to contain CR/LF or ":". This will not be checked by the REQUEST DOCUMENT statement. For valid header names, please see the HTTP specifications. For compatibility with the web interface, header names can be written with "_" instead of "-". Internally "_" is replaced by"-".

Request-Method

The HTTP request types are automatically generated for the given parameters. It is possible to overwrite the generated settings for special purposes, e.g., a multipart form-data upload.

The following values are supported:

- HEAD
- POST
- GET
- PUT

Request-Method	Operand	HEAD	POST	GET	PUT
WITH HEADER	4/5	optional	optional	optional	optional
WITH DATA	7/8	not specified	has to be specified	not specified	only with option ALL (operand 6)
RETURN HEADER	9 to 11	has to be specified	optional	optional	optional
RETURN PAGE	12	not specified	has to be specified	has to be specified	optional

Content-Type

If the request method is POST, a content-type header has to be delivered with the HTTP request. If no content-type is set explicitly, the following header is generated:

```
Content-Type: application/x-www-form-urlencoded
```

If you set up an own Content-Type (e.g., for multipart form-data upload), the following setting will be used:

```
multipart/form-data; boundary=----------7d127f331f0276
```

Wrong setting of the Content-Type can cause invalid HTTP requests, which will lead to unexpected errors.

Header Values Operand5

Header values are not allowed to contain CR/LF. This will not be checked by the REQUEST DOCUMENT statement. For valid header values and formats, please see the HTTP specifications.

operand6

Operand6 is a complete document that will be sent. This value is needed for the HTTP request method PUT.

operand7/8

Operand7 is the name of a DATA variable sent with this request. This value is needed for the HTTP request method POST (URL-decoding necessary, especially "&", "=", "%").

Operand8 is the value of a DATA variable sent with this request. This value is needed for HTTP request method POST (URL-decoding necessary, especially "&", "=", "%").

Note:
Operand7 and operand8 can only be used in conjunction with each other.

Restriction

If operand 7/8 is given, and the communication is "http://" or "https://" by default, the request method **POST** (see table above) with Content-Type **application/x-www-form-urlencoded** is used. During the request, the operands 7/8 will be separated by "=" and "&" characters. Therefore the operands are not allowed to contain "=" and "&", and because of URL-encoding, "%" characters. These characters are considered "unsafe" and need to be encoded as:

```
% (%25) & (%26) = (%3D)
```

Note: URL-encoding
When sending POST data with the content type **application/x-www-form-urlencoded**, certain characters must be represented by means of URL-encoding, which means substituting the character with %hexadecimal-character-code. The full details of when and why URL-encoding is necessary, are discussed in RFC 1630, RFC 1738 and RFC 1808. Some basic details are given here. All non-ASCII characters (i.e., valid ISO 8859/1 characters that are not also ASCII characters) must be URL-encoded, e.g., the file *köln.html* would appear in an URL as *k%F6ln.html*.

Some characters are considered to be "unsafe" when web pages are requested by e-mail.
These are:

- the tab character (%09)
- the space character (%20)
- [(%5B)
- \ (%5C)
-] (%5D)
- ^ (%5E)
- ‘ (%60)
- { (%7B)
- | (%7C)
- } (%7D)
- ~ (%7E)

When writing URLs, you should URL-encode these characters.

Some characters have special meanings in URLs, such as the colon (:) that separates the URL scheme from the rest of the URL, the // that indicates that the URL conforms to the Common Internet Scheme syntax and the percent sign (%). Generally, when these characters appear as parts of file names, they must be URL-encoded to distinguish them from their special meaning in URLs (this is a simplification, read the RFCs for the full details).
These characters are:

- " (%22)
- # (%23)
- % (%25)
- & (%26)
- + (%2B)
- , (%2C)
- / (%2F)
- : (%3A)
- < (%3C)
- = (%3D)
- > (%3E)
- ? (%3F)
- @ (%40)

operand9

Operand9 represents ALL received header values as a name value pair. The names and values are separated by ":" and "CR". (Internally, all "CR-NL"s will be transformed to "CR"s).

Restrictions

Handling New Line for HEADER ALL Operand9

HEADER ALL contains all headers delivered with an HTTP response. The first line contains the status, all following lines pairs of names and values. Names always end with a ":" and the value ends with the end of the line. If the HEADER ALL line contains CR/LF, all CRs are deleted, because some Unix servers do not deliver CRs.

operand10/11

Operand10 is the name of a HEADER received with this request. The HEADER is needed for HTTP.

Operand11 is the value of a HEADER received with this request. The HEADER is needed for HTTP.

Note:
Operand10 and operand11 can only be used in conjunction with each other.

Note: Return Header Name for Operand10
For compatibility with the web interface, header names can be written with "_" instead of "-". Internally, "_" is replaced by "-".

Return Header Name " "

At the beginning of an HTTP response, header status information about the request is delivered.

```
HTTP/1.0 200 OK
```

To provide this data to the Natural program without parsing "RETURN HEADER ALL ", the following special handling has been introduced:
If *operand* is a blank string, the status string is returned.

operand12

Operand12 is the document returned for this request.

operand13

Operand13 is the response (e.g. "HTTP 200 OK" or only "200")

Overview of Response Values Operand13

The following response values are common to HTTP/HTTPs:

Status	Value	Response
STATUS CONTINUE	100	OK to continue with request
STATUS SWITCH_PROTOCOLS	101	Server has switched protocols in upgrade header
STATUS OK	200	Request completed
STATUS CREATED	201	Object created, reason = new URL
STATUS ACCEPTED	202	Async completion (TBS)
STATUS PARTIAL	203	Partial completion
STATUS NO_CONTENT	204	No info to return
STATUS RESET_CONTENT	205	Request completed, but clear form
STATUS PARTIAL_CONTENT	206	Partial GET fulfilled
STATUS AMBIGUOUS	300	Server could not decide what to return
STATUS MOVED	301	Object permanently moved
STATUS REDIRECT	302	Object temporarily moved
STATUS REDIRECT_METHOD	303	Redirection w/o new access method
STATUS NOT_MODIFIED	304	If-modified-since was not modified
STATUS USE_PROXY	305	Redirection to proxy, location header specifies proxy to use
STATUS REDIRECT_KEEP_VERB	307	HTTP/1.1: keep same verb
STATUS BAD_REQUEST	400	Invalid syntax
STATUS DENIED	401	Access denied
STATUS PAYMENT_REQ	402	Payment required
STATUS FORBIDDEN	403	Request forbidden
STATUS NOT_FOUND	404	Object not found
STATUS BAD_METHOD	405	Method is not allowed
STATUS NONE_ACCEPTABLE	406	No response acceptable to client found
STATUS PROXY_AUTH_REQ	407	Proxy authentication required
STATUS REQUEST_TIMEOUT	408	Server timed out waiting for request
STATUS CONFLICT	409	User should resubmit with more info
STATUS GONE	410	The resource is no longer available
STATUS LENGTH_REQUIRED	411	The server refused to accept request w/o a length
STATUS PRECOND_FAILED	412	Precondition given in request failed
STATUS REQUEST_TOO_LARGE	413	Request entity was too large
STATUS URL_TOO_LONG	414	Request URL too long
STATUS UNSUPPORTED_MEDIA	415	Unsupported media type
STATUS SERVER_ERROR	500	Internal server error
STATUS NOT_SUPPORTED	501	"Required" not supported
STATUS BAD_GATEWAY	502	Error response received from gateway
STATUS SERVICE_UNAVAIL	503	Temporarily overloaded
STATUS GATEWAY_TIMEOUT	504	Timed out waiting for gateway
STATUS VERSION_NOT_SUP	505	HTTP version not supported

Restrictions

Response 301 - 303 (Redirection)

Redirection means that the requested URL has moved. As a response, a RETURN HEADER with the name LOCATION will be displayed. This header contains the URL where the requested page has moved to. A new REQUEST DOCUMENT request can be used to retrieve the page moved.

HTTP Browsers redirect automatically to the new URL, but the REQUEST DOCUMENT statement does not handle redirection automatically.

Response 401 (Denied)

The response "Denied" means that the requested page can only be accessed if a valid UserID and Password are provided with the request. As a response, a RETURN HEADER WWW-AUTHENTICATE will be delivered with the realm needed for this request.

HTTP browsers normally display a UserID password dialog, but with the REQUEST DOCUMENT statement, no dialog is displayed.

operand14

Operand14 is the runtime error if the request could not be performed.

Examples:

General Request

```
REQUEST DOCUMENT FROM "http://bolsap1:5555/invoke/sap.demo/handle_RFC_XML_POST" WITH USER
   DATA
  'XMLData'     QUERYXML
  'repServerName'    ' NT2'
     RETURN PAGE RESULTXML
```

Simple Get Request (no data)

```
 REQUEST DOCUMENT FROM "http://pcnatweb:8080" WITH
    RETURN PAGE RESULTXML
    RESPONSE rc
```

Simple Head Request (no return page)

```
REQUEST DOCUMENT FROM "http://pcnatweb" WITH
    RESPONSE rc
```

Simple Post Request (default)


```
REQUEST DOCUMENT FROM "http://pcnatweb/cgi-bin/nwwcgi.exe/sysweb/nat-env" WITH
    DATA
  'XMLData'         QUERYXML
  'repServerName'   'NT2'
    RETURN PAGE RESULTXML
    RESPONSE rc
```

Simple Put Request (with data all)

```
REQUEST DOCUMENT FROM "http://pcnatweb/test.txt" WITH
    DATA ALL  #document
    RETURN PAGE RESULTXML
    RESPONSE rc
```

